Developing Use Cases from Organizational Modeling

Victor F.A. Santander, Jaelson F. B. Castro
Universidade Federal de Pernambuco – Centro de Informática
{vfas,jbc}@cin.ufpe.br

Abstract: The object oriented development paradigm has attracted many supporters in the Software Engineering community. One of the most important advances was the Unified Language Modeling (UML), a standard for visual modeling. Use Cases Diagrams have been used for capturing system functional requirements. However, the system development occurs in a context where organization processes are well established. Therefore, we need to capture organizational requirements to define how the system fulfils the organization goals, why it is necessary, what are the possible alternatives, what are the implications to the involved parts, etc. Unfortunately, UML and other scenario-based techniques are ill equipped for modeling organizational requirements. We need other techniques, such as i*, to represent these aspects. Nevertheless, organizational requirements must be related to functional requirements represented as Use Cases. In this paper we present some guidelines to assist requirement engineers in the development of use cases from the organizational models represented by i* technique.

Keywords: Scenarios, Organizational Modeling, Requirement Engineering.

1. Introduction

The development of complex software systems, susceptive to certification and within a limited budget, has been a constant challenge for software engineers. Several techniques, methodologies and tools have been proposed to assist and support the development of quality software [9] [12] [16] [26]. The software crisis [3] remains present in many aspects. Frequently, we can find software that does not meet the real needs of the customers. Among the main reasons for this failure we can emphasize: the lack of well defined software processes, software requirements not well understood and agreed, use of unsuitable techniques as well as the own nature of the complex software.

The software engineering community has emphasized Requirement Engineering as the most critical activity in the software development process. System requirements must be appropriately elicited, analyzed, validated and managed to meet the stakeholders needs. Requirement documents usually are incomplete or inconsistent, generating software products with low quality and unsatisfied customers. Therefore, it is very important to perform the requirement engineering activities efficiently as well as to try to eliminate requirement problems still in the early phases of the software development [26]. Current researches in requirement engineering have proposed techniques and methodologies, which aim at the development of more complete and consistent requirement documents. Among the main aspects that must be considered in the elaboration of the requirement documents we can mention: to consider all relevant elements of the organizational environment where the software will execute, to elicit the functional and non-functional requirements, to assure that these requirements are the most complete as possible as well as to assure that the documented requirements meet the needs of users and customers.

To define system requirements we can use many techniques presented in literature. In the 1970's, techniques such as DFD and ER [13] which were developed to elicit and document functional requirements. More recently, techniques such as scenarios, use cases, ethnography, formal methods have been considered as viable alternatives for requirement engineers. Among these techniques, we can highlight scenario-based techniques. Scenarios are used to describe the interactions between users and software systems. One of the most popular scenarios technique are the so-called Use Cases. As a consequence Use Cases have been incorporated into the Unified Modeling Language (UML), a standard for visual modeling, one of the most important advance in the object-oriented development [2]. In UML, Use Cases are used for capturing system functional requirements.

However, system development occurs in a context where organizational processes [4] [5] [27] are well established. Therefore, we need to capture organizational requirements to define how the system fulfils the organizational goals, why it is necessary, what are the possible alternatives, what are the implications to the involved parts, etc. Unfortunately, use cases in UML and other scenario-based techniques are ill equipped for organizational requirement modeling. We need others techniques, such as i* [27] to represent these aspects. We argue that i* framework, is well suited to represent organizational requirements that occur during the early-phase requirements capture, since it provides adequate representation of alternatives, and offers primitive modeling concepts such as softgoal and goal.

Nevertheless, organizational requirements must be related to functional requirements represented with techniques like Use Cases. Usually, use case development demand great experience of the requirement engineers. The heuristics presented in the literature to develop use cases are not sufficient to permit a systematic development. Indeed, they do not consider relevant organizational aspects such as goals and softgoal. In this work, we propose some guidelines to support the integration of i* and use case technique. We describe some heuristics to assist requirement engineers to develop Use Cases in UML based on the organizational i* models. The i* framework can be used to improve the understanding of how systems fulfils organization goals. In the figure 1, we present elements of our proposal.

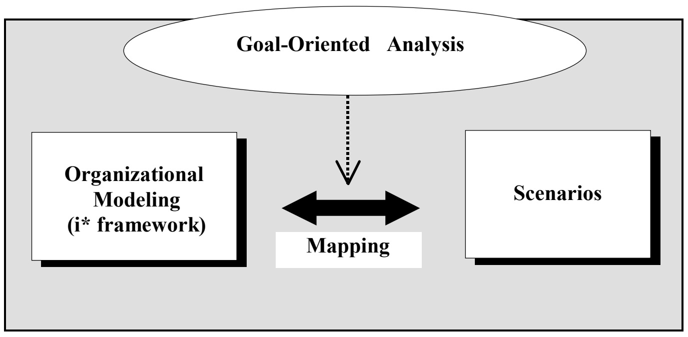

Figure 1. A vision of mapping process between organizational modeling and scenarios.

The objective is to integrate in fact, goal driven methodologies, the i* framework and scenarios, indicated under a goal-oriented analysis. There are very promising early results researches such as [6] [18] [19] [20] [23]. In our proposal, elements such as goal, softgoal, resource and task presented in i* are analyzed to generate goals of use cases. In the i* framework, actors depend each other to reach goals and softgoal and to realize tasks. We must also investigate which and how these elements are related to system requirements and system use cases.

In this paper we present guidelines that allow requirement engineers to develop use cases (and associated scenarios) from organizational modeling described in i* framework. This work is an evolution of the [24]. The original guidelines presented are improved as well as better described into three well-defined phases (section 4, figure 5). These phases provide a more systematic way to integrate the i* technique and use cases. Besides, other guidelines (such as Guidelines 1.5 and 2 - 2º Proposal Step in section 4) are introduced to consider important aspects such as the use of the goal oriented analysis in the i* and use case integration process. This approach is used in our proposal to aid requirements engineers to discover and define use case goals from i*. These goals are classified into three levels (business, contextual or user goal). The analysis of these goals can aid requirements engineers to discover other use cases as well as assure that the goals associated with use cases are in an appropriate abstraction level. Other evolution related with our work are the evolution of the i* models and the possible dependences modifications between actors that can to originate other use cases for the intended software system. In the section 4 are described all guidelines of our proposal.

This paper is organized as follows. Section 2 introduces the concepts used by i* framework to represent organizational requirements and early requirements. In Section 3, we present the main concepts of scenarios-based techniques, especially use cases technique. Section 4, we describe guidelines and steps to integrate i* organizational models and Use Cases diagrams. In order to show the viability of our proposal we also describe how these guidelines are used in association with the meeting scheduler problem. Section 5 concludes the paper with a summary of its contribution.

2. The i* Modeling Framework

When developing systems, we usually need to have a broad understanding of the organizational environment and goals. The i* framework [27] provides understanding of the reasons ("Why") that underlie system requirements. This technique offers a modeling framework that focuses on strategic actor relationships. Usually, when we try to understand an organization, the information captured by standard modeling techniques such as DFD, ER, Statechart, etc, focuses on entities, functions and flows, states and the like. They are not capable of expressing the reason (the "why's") of the process (motivations, intentions, rationales). The ontology of i* caters to some of these more advanced concepts. Thereby, i* allows the description of the intentions and motivations involving actors in an organizational environment. It offers two models to represent these aspects: The Strategic Dependency (SD) Model and the Rationale Dependency (SR) Model. The i* technique have been used in various application areas such as: requirement engineering (like this work), business process reengineering, organizational impact analysis, software process modeling.

2.1 The Strategic Dependency Model

This model focuses on the intentional relationships among organizational actors. It consists of a set of nodes and links connecting them, where nodes represent actors and each link represents the dependency between two actors. The depending actor is called Depender and the actor who is depended upon is called Dependee. Hence, this model consists of a set of relationships among actors, capturing intentions and motivations among them. The i* framework defines four types of dependencies among actors: goal dependency, resource dependency, task dependency and softgoal dependency. Each one of these dependencies represents different intentions and motivations in the organizational environment. In a Goal Dependency, an actor depends on another to fulfil a goal, without worrying how this goal will be achieved. In a resource dependency, an actor depends on another to provide a physical resource or information. In a task dependency, an actor depends on another to realize some sequence of activities. Finally, in a softgoal dependency an actor depends on another to fulfil a fuzzy goal. The softgoal dependency is a different dependency because it represents a goal not precisely defined. In requirement engineering, a softgoal represents non-functional requirements.

In i* we can also define different degrees of dependencies between actors: open, committed or critical. Actors can be refined into agents, roles and positions. An agent is an actor with concrete physical manifestations (person or system). A role is an abstract characterization of the behavior of a social actor within some specialized context, domain or endeavor. A position is a set of roles typically played by one agent.

Figure 2 presents the Strategic Dependency (SD) Model describing a Meeting Scheduler [27]. The meeting initiator actor possesses a set of dependencies with the meeting participant actor. The resources dependencies ExclusionDates(p) and PreferredDates(p) indicate that the participant should supply information about unsuitable dates as well as some favorite dates. The Meeting Initiator depends on these resources to continue the scheduling process.

On the other hand, the Meeting Participant depends on a proposed date (ProposedDate(m)) supplied by the Meeting Initiator that might agree or not with the date. The goal dependency AttendsMeeting(p,m) considers that Meeting Participant should satisfy the goal to attend the meeting. In this situation, the Meeting Participant actor can choose the way that this goal will be fulfilled. The Agreement(m,p) dependency represents a resource that a Meeting Participant should supply to show whether one participant do not agree with the proposed date. The Assured(AttendsMeeting(ip,m)) softgoal dependency is a reference to a softgoal with a subjective evaluation.

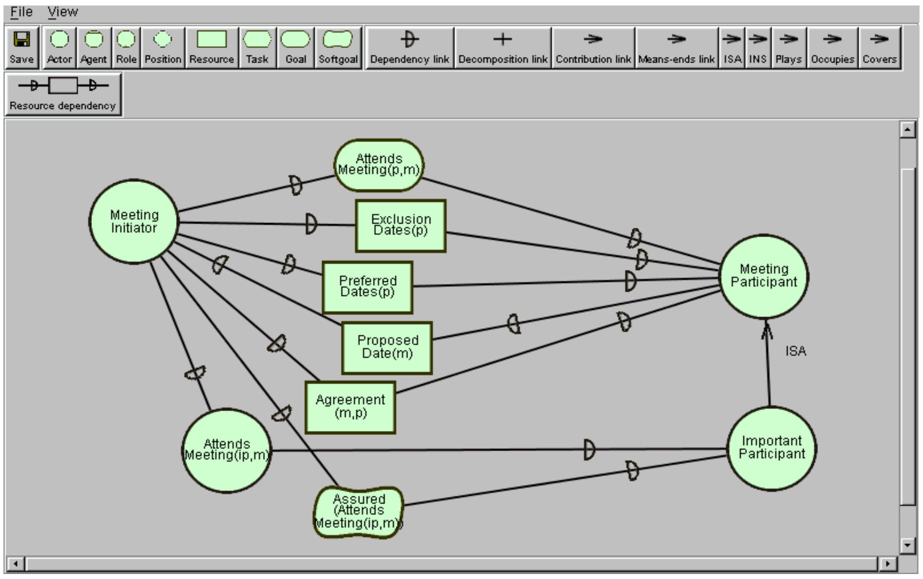

Figure 2. Strategic Dependency Model for the Meeting Scheduler Problem.

2.2 The Strategic Rationale Model

The Strategic Rationale (SR) Model is a supplementary model to the Strategic Dependency (SD) Model. This model allows modeling of the reasons associated with each actor and their dependencies. Each actor externally possesses dependencies with other actors but internally it possesses goals and routines that impel and justify each dependency. To develop this model we must investigate the reasons associated with each actor in relation to the dependencies with other actors. A good way to begin the decomposition is to observe how the Dependee actor can satisfy the Dependum associated with the same, and then to investigate and decompose intentions and strategic organizational reasons as a whole. Nodes and links also compose this model. The nodes have as base the types of dependencies defined in the SD model: **goal, resource, task and softgoal**. The links can represent two types of connections: **means-ends and task-decomposition**. These links are described as:

- Link means-ends: This link is associated to the obtaining of a certain end, which can be a goal, resource, softgoal or task. The mean to obtain the end is defined as tasks that are necessary for reach the end.
- Link task-decomposition: A node task is linked to their component tasks through a decomposition connection. The four types of nodes existent can be linked, decomposing a task through this link type. It is permitted by linking the decomposition into smaller units of a node task to represent the way and the reasons associated with the accomplishment of the task.

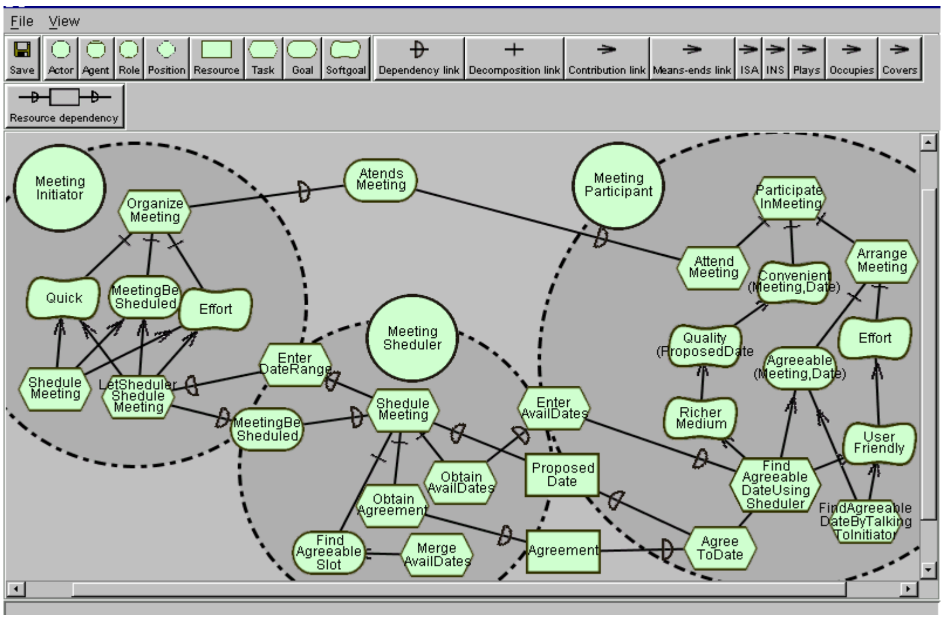

Figure 3. Strategic Rationale (SR) Model to the Meeting Scheduler System.

In figure 3, we present an example of the Strategic Rationale (SD) model. This model represents the **reasons** associated with actors in the organizational environment to schedule a meeting. The Meeting Scheduler actor, which represents a software system, performs partially the meeting scheduling. The other two actors presented in this model, the Meeting Initiator and the Meeting Participant, are responsible for providing or receiving information to the system. The Meeting Scheduler actor possesses a Schedule Meeting task which is decomposed into three other tasks using the **task-decomposition relationship**: *ObtainAvailDates, FindAgreeableSlot* and *Obtain Agreement*. These tasks represent the work that will be accomplished by the meeting scheduler system. In the same way, the other actors of the organizational model are decomposed to represent the internal reasons that satisfy the dependencies with other actors.

3. Use Cases in UML

The scenarios-based techniques have been used in the software engineering to understand, model and validate users requirements. Some approaches proposing the use of scenarios to elicit and validate requirements includes [23] [17] [16]. Among these techniques, use cases have been receiving a special attention. Use cases diagrams are used in the Unified Language Modeling (UML), a standard for visual modeling which has been one of the most important advances in the object oriented development paradigm.

Use Cases in UML [2] are used to describe the using of a system by actors. An actor is any external element that interacts with the system. A use case describes the sequence that a user accomplishes when it interacts with a system to accomplish a task or goal. However, the description of Use Case does not treat how this interaction will be implemented. Subsequent phases in the software engineering such as Design and Implementation will focus on this aspect.

A single use case can generate several scenarios. Scenarios are for use cases like instances are for classes, meaning that a scenario is a use case's instance. The use of the system by the actor can involve several ways depending on the execution context of the system. These ways are the possible use cases scenarios. The basic way to accomplish a use case, without problems or mistakes is called primary scenario. In this scenario, the execution of the steps to accomplish the basic functionality of the uses case is obtained with success. On the other hand, alternative ways as well as mistaken situations can be represented through secondary scenarios.

The secondary scenarios describe possible alternatives sequences in a primary scenario. Secondary scenarios can be described separately or as extension of a primary scenario. If a secondary scenario is more complex and includes several steps, it is convenient to describe it separately. It is important to notice that different definitions of scenarios are also possible. For instance, in [17] the scope attributed to scenarios is wider. In that approach, scenarios are descriptions of situations in an evolving environment. Scenarios evolved together with the software development process, leaving initially the description of the macrosystem. Besides, scenarios are naturally linked to a LEL (Language Extended Lexicon) and a BMV (Basic Model View) of the requirements baseline [17].

UML also allows the structuring of use cases. The relationships in UML include:

- **<< include >>:** When a group of the common steps to several use cases is detected in the system, we can join these steps in a single use case, which can be used by others use cases. Thereby, a specific use case can refrain a common behavior to several use cases, establishing that others use cases can use the common use case (i.e. to include it) when necessary.
- **<<extend >>:** we used this relationship for an optional or conditional sequence of steps that we want to include in one use case. This sequence of steps should be described in one specific use case that could be used by others use cases when necessary. The use of the extended use case happens in an optional or conditional behavior situation.

- **<<generalization>>:** generalization among use case is similar to the classes generalization in the object-oriented paradigm. This means that one "son" use case inherits the behavior and structure of the "father" use case. The "son" use case is a specialization of the "father" use case adding new structure and behavior as well as modifying the behavior of the "father" use case.

We can also use the generalization mechanism among use cases to relate actors. The actor 2, for instance, can be a specialization of the actor 1. In this situation, the actor 2 inherits the whole structure and behavior of actor 1 and can add new structure and behavior for the actor 1. Other additional mechanisms proposed in UML to represent and describe use cases are presented in [2]. The figure 4 presents use cases notations in UML.

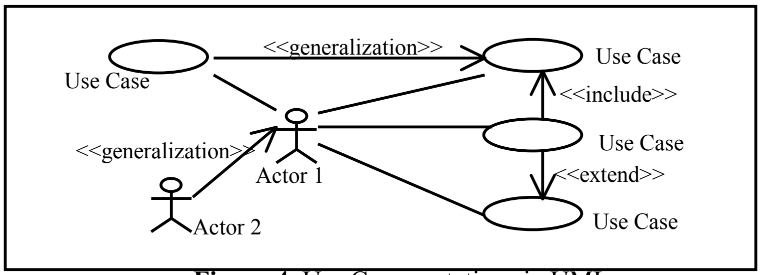

Figure 4. Use Cases notations in UML.

The use cases development process begins with the discovery of the system actors and follows with the use cases discovery associated with these actors. For each actor, we find all the related use case.

The second step consists of defining the basic way (primary scenario) and later the alternative way (secondary scenarios) for each use case discovered. The third step involves revising descriptions of the use cases behavior aspects, finding the relationships << include >>, << extend >> and << generalization >>. This process is usually iterative and incremental [15]. After defining all the use cases and actors of the system, use case diagrams are developed using the notations presented in the figure 4.

Several heuristics to aid requirement engineers in the use case development are presented in [2] [25]. However, the discovery and description of use cases is not so simple, because, in most of the situations, it demands a certain degree of experience from requirement engineers. The first aspect is discovering which actors really wish to reach some service or functionality from the system.

Another difficulty, is defining which use cases really satisfy the actors interests, providing them with a relevant result. One of the alternatives to assist this process is indicated in [10]. Cockburn argue that stakeholders possess goals in relation to the system to be developed. These goals can be of higher or lower level and can originate use cases for the intended system.

However, the main challenge is related to how system goals can be discovered initially. Traditionally, requirement engineers accomplish this work, using mainly their experiences to discover these goals and then to describe use cases. In this sense, we believe that a viable alternative is to begin the use case discovery process investigating the goals and other elements represented in the **i* organizational models**. The use of the defined information in organizational models can facilitate the use cases development as well as turn this process more systematic. We have been observing that the heuristics presented in the literature [2] [25] to use cases development, are not satisfactory and still turn the process ad-hoc. Besides, in most situations, use cases are developed without considering the organizational requirements previously defined by the organization. Thereby, in this paper we propose to develop use cases starting from the observation and analysis of organizational goals and other elements represented through **i***.

4. Developing Use Cases from Organizational Modeling.

We have shown that i* provides an early understanding of the organizational relationships in a business domain. As far as we continue the development process, we need to focus on the functional and non-functional requirements of the system-to-be, which will support the chosen alternative among those considered during early requirements. As first step in the late requirements phase we can adopt use cases to describe functional requirements of the system. Traditionally, use cases in UML are developed without efficiently considering organizational requirements previously defined. We argue that the use cases development from organizational modeling using i* allows that requirement engineers to establish a relationship between the functional requirements of the intended system and the organizational goals previously defined in the organization modeling. Besides, through a goal-oriented analysis of the organizational models, we can derive and map goals, intentions and motivations of organizational actors to main goals of use cases. We assumed, that for each use case we have associated a main goal, which represents what the user aims to reach as a result of the execution of the use case. In our proposal, the use cases scenarios description can begin investigating the organizational models, which are well known and understood by all of the stakeholders. This can avoid future inconsistencies, ambiguities and uncertainties regarding to the intended system requirements.

To guide the mapping and integrating process of i* organizational models and use cases, we have defined some guidelines. These guidelines must be applied following the steps represented in the figure 5. In this figure, steps 1, 2 and 3 represent the activities to discover system actors, use cases for the actors and scenarios for these use cases. The input for the integration process (connected rectangle with the step 1), is composed by the Strategic Dependency (SD) and Strategic Rationale (SR) models developed through i* framework. In steps 1 and 2, the input is the Strategic Dependence (SD) Model. The description of scenarios for use cases (step 3) can be obtained from elements represented in the Strategic Rationale (SR) Model. The results of the integration processes (connected rectangle as output of step 3) are use cases diagram for the intended system and scenario textual descriptions for each use case. For each step, we describe guidelines to aid requirement engineers in the accomplishment of the step. These guidelines help the requirement engineers in the accomplishment of a goal-oriented analysis of the SD and SR models, aiming to derive and refine goals for use cases.

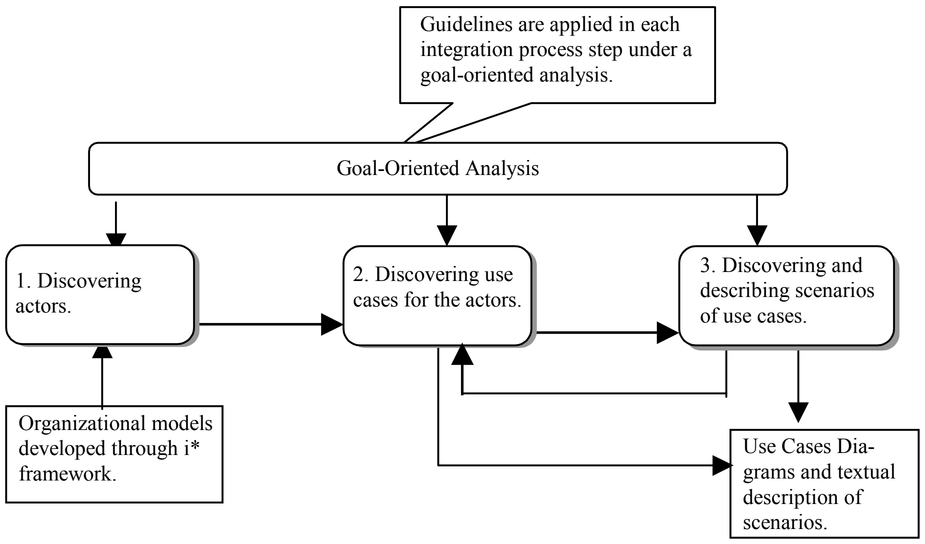
Guidelines are applied in each integration process step under a goal-oriented analysis.
Goal-Oriented Analysis
1. Discovering actors.
2. Discovering use cases for the actors.
3. Discovering and describing scenarios of use cases.
Organizational models developed through i* framework.
Use Cases Diagrams and textual description of scenarios.

Figure 5. Steps of the integration process between i* and use cases in UML

In the sequel we suggest heuristics for the use cases development from organizational modeling with i*.

1º Proposal Step: Discovering System Actors.

Guideline 1: ***every actor in i**** should be analyzed for a possible mapping for ***actor in use cases***; *For instance, we can analyze the Meeting Participant actor in figure 2.*

Guideline 2: initially, the actor analyzed in i * should be external to the intended system. Actors in use cases cannot be the software system. The Meeting Participant actor is external to the system because it will interact with the intended system to meeting schedule.

Guideline 3: if the actor is external to the system, it should be guaranteed that the actor in i* is an actor candidate in the Use Case diagram. For this purpose, the following analysis is necessary:

> *Guideline 3.1:* the actor dependencies in i* must be relevant at the point of view of the intended system; For instance, the Meeting Participant actor in i* can be mapped to use case actor, considering that dependencies associated with it, characterize it as important in an interaction context with the system. The dependencies between Meeting Participant and Meeting Initiator (see figure 2), show that Meeting Participant has the responsibility to attend and supply information that will be treated in a meeting scheduler system.

Guideline 4: actors in i*, related through the IS-A mechanism in the organizational models and mapped individually for actors in use cases (applying guidelines 1, 2 and 3), will be related in the use case diagrams through the <<generalization>> relationship. For instance, the IS-A relationship between Meeting Participant and Important Participant in figure 2, can be mapped to generalization relationship among these actors in the use case diagram.

2º Proposal Step: Discovering Use Cases for the Actors.

Guideline 1: for each discovered ***actor*** for the system (step 1), we should observe all their dependencies (dependum) of point of view as ***dependee***, looking for use cases for the actor; For instance, some use cases can be associated with the ***Meeting Participant*** actor observing their dependencies presented in i*:

> *Guideline 1.1*: the goal dependencies associated with the actor in i* should be evaluated; usually, the most goals in i* can be mapped to use cases goals; For instance, in the figure 2, the goal dependency AttendsMeeting(p,m) between Meeting Initiator and Meeting Participant can be mapped to the ***AttendsMeeting*** use case, which will contain the several steps accomplished by Meeting Participant to attends to the meeting.
>
> *Guideline 1.2*: the resources dependencies associated with the actor should be evaluated; the most important question in this situation is: if an actor depends on another actor for obtaining of a resource(s), which is the ***goal*** of the obtaining of that resource(s)? If there is a more abstract goal, probably this goal will be candidate to be the goal of the use case for the actor. For instance, observing the Meeting Participant as Dependee, we have two resources dependencies ExclusionDates(p) and PreferredDates(p). These resources can be mapped to the DefineAvailDates use case for the MeetingParticipant actor, which will contain the steps to define possible meeting scheduling dates. We understand that the main goal obtaining these resources is the definition of available dates for the meeting by the participants.
>
> *Guideline 1.3:* the task dependencies associated with the actor should be evaluated; the key question of this situation is: if an actor depends on another actor in the accomplishment of a task, it should be investigated if this task is involved in a group of tasks that aim at a more abstract goal for the actor. Which would that aim be at?. For instance, in the figure 3, the task dependency Organize Meeting associated with the Meeting Initiator actor, can be mapped for the use case Organize Meeting for the Meeting Initiator actor, representing the steps of the system used to organize a meeting.
>
> *Guideline 1.4:* the sofgoal dependencies associated with the actor should be evaluated; Typically, the sofgoal dependency in i* is a non-functional requirement for the intended system. Thereby, a soft-

goal do not represent a use case of the system but a non-functional requirement associated with a use case of the system. For instance, the softgoal **Assured(AttendsMeeting(ip,m))** between Meeting Initiator and Important Participant actors can be mapped into a non functional requirement associated with the use case AttendsMeeting. This non-functional requirement indicates that it is necessary to assure that the Important Participant attends to the meeting.

Guideline 1.5: consider i* model evolution to detect changes in the dependency relationships between actors. These changes can represent new use cases for the system. For instance, initially (figure 2) for the resource Agreement(m,p), the depender in the relationship is the Meeting Initiator actor. In figure 6, we are now dealing with late requirements, i.e. the software to be (Meeting Scheduler) was introduced. Hence, the depender has changed to Meeting Scheduler actor, which represents a meeting scheduler system. This means that in the illustration 6, the Meeting Scheduler actor depends on the MeetingParticipant actor to agree with a date proposal for the meeting. In contrast, in the manual process of meeting scheduling presented in figure 2, who depends on an agreement in relation to a proposed date for the meeting, is the Meeting Initiator actor. What is important in this situation is to observe the objective of the goal with the obtaining of the Agreement(m,p) resource. So, we can define a use case ***Agreement,*** which will include several necessary steps by the Meeting Participant to agree with the proposed date.

Guideline 2: to analyze the special situation, where an actor and their use cases are discovered following the step 1 of the proposal, possesses dependencies in relation to an actor in i* that represents an intended software system or part of it. These dependencies can generate use cases. For instance, the goal dependency MeetingBeScheduled between Meeting Initiator and Meeting Scheduler system in the figure 6, point out for the definition of the use case ScheduleMeeting for the Meeting Initiator actor, which represents the use of the system by the actor, describing the details of the meeting scheduling process.

Guideline 3: to classify each use case and its main goal associate, in a level of proposed goal (business, contextual or user goal). This classification aids in the identification of possible new use cases. A business goal represents a high level intention, related with business processes, that the organization or user possesses in the context of the organizational environment. An example could be the goal "organizing a meeting in the possible shortest time". A context goal represents an alternative for the satisfaction of a business goal". An example could be the goal, "meeting scheduling by software system". Finally, obtaining a user goal results in the direct discovery of a relevant functionality and value for the organization actor using a software system. An example could be the goal, "the meeting participant wish to attend the meeting"

3º Proposal Step: Discovering and Describing Scenarios of Use Cases.

Guideline 1: to analyze each actor and their relationships in the Strategic Rationale (SR) model, to extract information that can lead to the description of the use cases scenarios for the actor. For instance, let us observe the Strategic Rationale (SR) Model in figure 3. From the Meeting Initiator actor point of view, we verify that the Schedule Meeting task is decomposed into ObtainAvailDates, FindSuitableSlot and ObtainAgreement. Since the software system objective is to accomplish meeting scheduling, we could adopt that these tasks are the necessary high-level steps to accomplish a meeting schedule. Therefore, the use case called ***Schedule Meeting*** could be defined for the Meeting Initiator actor. This use case could contain the steps (the primary scenario description) regarding the need to obtain from each Meeting Participant, the available dates for a meeting (ObtainAvailDates); from of participants readiness to find the meeting dates that could be scheduled (FindSuitableSlot); and to obtain the participants agreement for a proposed meeting date (ObtainAgreement).

In order to improve the comprehension about these guidelines, we have applied them to the meeting scheduler problem. Recall figure 2 described a Strategic dependency (SD) model for the meeting scheduler, which do not consider the existence of the Meeting Scheduler software system. Additionally, we also consider in our case study (see figure 6) an evolution of this early model, by considering the existence of the actor representing the meeting scheduler software. We also consider the model in figure 3, which represents the Strategic Rationale (SR) Model associated with the Strategic Dependency (SD) model in the figure 5. These three models compose the organizational models used to discover and describe use cases in UML for the Meeting Scheduler system.

Following the steps proposed in the figure 5 and applying the appropriated guidelines, we have:

- Observing figure 6, we can find actors candidates for the use case development. Following the defined guidelines in the ***1st step*** of the proposal, we verified that one of the analyzed actors does not follow guideline 2. The Meeting Scheduler actor is a system, which is the software itself that we aimed at to develop. Therefore, this i* actor cannot be considered as use cases actor. The other i* actors are considered relevant because their strategic dependencies refer to relevant aspects for the meeting scheduler system (guideline 3) development. So, the list of candidates use cases actors includes: ***Meeting Initiator, Meeting Participant and Important Participant***. Better observing these candidates, we verify that Important Participant is a type (relationship IS-A) of participant. According to guideline 4 (1st step), we consider this actor a specialization of Meeting Participant. Then we establish a generalization relationship among these actors, in agreement with the defined notations in the Use cases diagrams in UML [2].

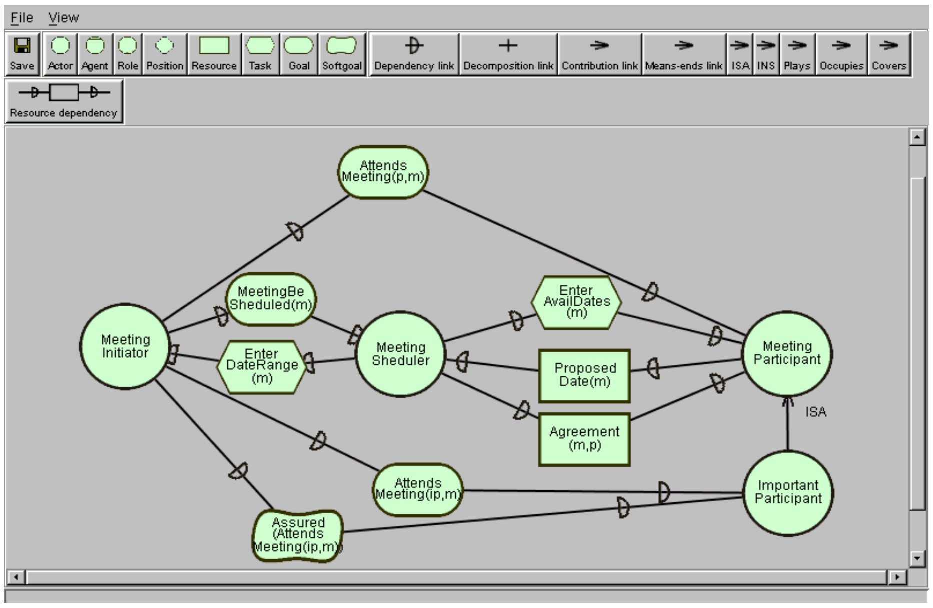

Figure 6. Strategic Dependency Model for the meeting scheduling with computer-based scheduler.

Continuing in the use cases discovery from i* organizational models, in agreement with the defined process in the figure 5, the next step is to discover and relate use cases for each actor. Following the guidelines presented in the ***2º Step*** *(Discovering Use Cases for the Actors*) in our proposal, we can verify:

- For the Meeting Participant actor, observing this actor as Dependee, we can indicate some use cases originated from the actors dependency relationships (guideline 1). Initially, we should observe the goals dependencies (guideline 1.1) of the actor as Dependee. In figure 2, we verify the goal AttendsMeeting(p,m), which represents the need of the meeting participant actor to attend the meeting. This goal originates the use case ***AttendsMeeting***. Several steps are necessary to achieve this goal. Typically, this is a user goal, considering the definition of objectives levels proposed in the guideline 3. The fulfillment of the use case goal brings a relevant result for Meeting Participant actor, allowing the same to attend to the meeting. Usually, the description of the primary scenario (to be accomplished later) for this use case, will present other users goals that can originate new use cases for the system.

  Continuing our analysis, we can observe in figure 2, associated with the Meeting Participant (Dependee) actor, three resource dependencies: ExclusionDates(p), PreferredDates(p) and Agreement(m,p). Following guideline 1.2, we conclude that the main goal of obtaining of Agreement(m,p) resource is an scheduled date agreement from each participant. We could consider that in this agreement process, each participant could agree with the proposed meeting date with certain schedule restrictions or duration time, for example. Still, the agreement could involve an analysis of other possible dates. In other words, to obtain the scheduled date agreement, firstly would involve several interactions steps between the system and the Meeting Participant actor that could be defined in one use case called **Agreement** for the Meeting Participant actor.

We also should analyze the resources dependencies ExclusionDates(p) and PreferredDates(p) (figure 2). Using the guideline 1.2 we can observe that the resources ExclusionDates(p), PreferredDates(p) are necessary to obtain a more abstract goal that is related to the definition of the available dates by the Meeting Participant actor. This goal originates the use case ***DefineAvailDates*** for the Meeting Participant actor.

- To find use cases candidates for the Meeting Initiator actor, we should follow the same guidelines (*2º Proposal Step)* for Meeting Participant actor. We have two dependencies associated with the Meeting Initiator actor. In figure 2, the resource dependency ProposedDate(m) between the actors Meeting Participant and Meeting Initiator, indicates that Meeting Initiator should provide a proposed date for the meeting scheduling. If we observe figure 6, we will notice that the dependency ProposedDate(m) was attributed to the Meeting Scheduler system. This resource will be originated from an operation accomplished internally in the system, with previously criteria definition and without interaction with the Meeting Initiator actor. Therefore, a meeting date will be proposed by the system, not being justified the definition of one use case for this purpose (see guideline 1.2 and 2).

  The other dependency associated with the Meeting Initiator actor is a task dependency EnterDateRange(m) (figure 6). Following the guideline 1.3, we can consider that the task of supplying a date range for the meeting scheduling is quite simple, because it just seeks to supply information defined previously for the Meeting Initiator actor. Thereby, probably the task EnterDateRange(m) will be a step in the description of the use case related with a Meeting Schedule goal. So, this dependency does not directly originate a use case to the Meeting Initiator.

  As there are no more dependencies for the Meeting Initiator as Dependee, the guideline 2 should be followed. Observing figure 6, we visualize the goal dependency MeetingBeScheduled between Meeting Initiator and Meeting Scheduler (which is the software system). For that, it is clear the interaction needing between the Meeting Initiator actor and the system for the accomplishment of the meeting scheduling. Therefore, we can define the use case ***Schedule Meeting*** that represents the use of the system by the Meeting Initiator actor. In this use case, we describe the details of the meeting schedule process.

Thereby, after we have used the proposed guidelines (*2º Proposal Step*), we have discovered the use case **ScheduleMeeting** for the Meeting Initiator actor and the use cases **DefineAvailDates, AttendsMeeting, and Agreement** for the Meeting Participant actor**.** Therefore**,** we can begin the description of the primary and secondary scenarios and the use cases relationships (*3º Proposal Step*). At this point, the Strategic Dependencies (SD) models and mainly the Strategic Rationale (SR) model, are used as source of information for the scenarios description and the use cases relationships.

For example, the use case ***Schedule Meeting*** discovered for the Meeting Initiator actor represents the use of the system by Meeting Initiator to accomplish the meeting scheduling. This use case should contain all the necessary steps to schedule a meeting that begins when the Meeting Initiator supplies information to the system such as the date range to schedule a meeting. Based on the supplied dates range by Meeting Initiator, the system must find available dates for all the participants for the meeting as well as elaborate a consensus dates list within which will be chosen a date to be proposed and agreed. This process must result in a consensus-scheduled date for the meeting and later in the confirmation of this date for all the participants. Thus, for the use case Schedule Meeting, we could have the primary scenario with the following steps:

Use Case: **Schedule Meeting**
Actor: **Meeting Initiator**
Use Case Goal: **Schedule a Meeting**
Primary Scenario:

1. The use case begins with the Meeting Initiator actor supplying the system with a date range for the meeting;
2. The system should request from participants (Meeting Participant) an available date list for the meeting based on the proposed date range by the Meeting Initiator; (the use case *DefineAvailDates* is included <<include>> in this step).
3. The system should find a consensus dates list, filtering information observing the available dates sent by the participants and the proposed date range sent by Meeting Initiator;
4. Based on the consensus list, the system defines a date to be scheduled for the meeting and it accomplishes the participants consultation;
5. The Meeting Initiator expects that the system requests the agreement for a scheduled meeting date, containing the meeting scheduling process when all participants agree with the date. (The use case *Agreement* is included << include >> in this step).

The information for the description of this use case has as main source the Strategic Rationale (SR) Model presented in the figure 3. The base information for the step

1 in this use case, is extracted from the task dependency EnterDateRange, establishing the need that Meeting Initiator supplies date range, within which should take place the meeting scheduling. We considered that the process of establishing a date range is quite simple, not justifying the creation of another use case for that purpose.

Steps 2 and 3 are extracted from the decompositions of the task Schedule Meeting (associated with Meeting Scheduler in the figure 3). Step 2 is extracted from the task ObtainAvailDates analysis. The use case DefineAvailDates is included << include >> because it represents the necessary steps for the acquisition of the available dates list by participants. Step 3 generates the observation of the goal FindAgreeableSlot and the task MergeAvailDates. This step represents the internal actions of the system to define a list of the consensus dates for the meeting scheduling.

Step 4, is extracted from observation of the resource dependency ProposedDate in relation to the task Schedule Meeting (figure 3). It is assumed, given the defined information in the models of the figures 2 and 6, that the proposed date should be defined by the system, previously using some established and defined criterion by the Meeting Initiator, taking as base for example, priorities of organization meetings. Step 5, derives from the system need to obtain the agreement for the chosen date for the meeting scheduling. This information arises from the observation of the task ObtainAgreement, associated with the resource Agreement(m,p) for Meeting Participant (figure 3). Previously, in the use case discovery for the system, we assumed that to obtain the goal that each participant agree with the proposed date, it was necessary the accomplishment of some interaction steps between each participant and the Meeting Scheduler. These steps should be described in the use case Agreement. For this reason, this use case is included <include> in the step 5.

We can describe the others use cases in a similar way, adopting processes and guidelines provided in this case study.

After we have applied the proposed guidelines to this case study, we can define, as described in the figure 7, a version of the use cases diagram in UML for the Meeting Scheduler system. The descriptions of the discovered use cases could still be modified or complemented, as new relationships are found. Another important aspect is that the development of other use cases depends on the requirement engineers' experience. However, modeling of this nature can vary, it aims to facilitate the understanding as well as to establish an agreement between customers and developers in the system requirements definition.

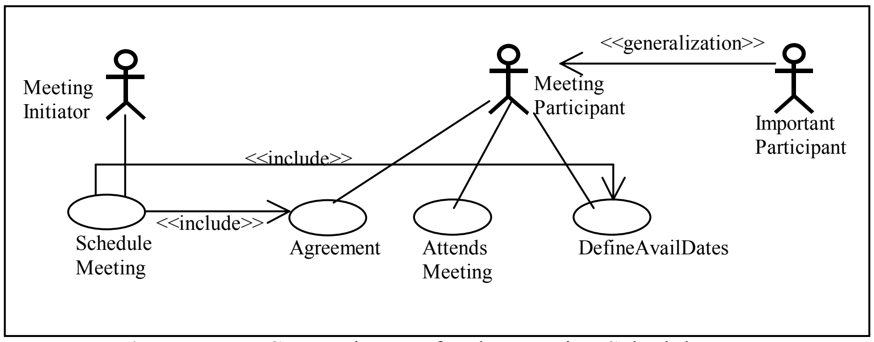

Figure 7. Use Cases Diagram for the Meeting Scheduler system.

Thereby, using the i* organizational models and following the proposed guidelines, we can extract a useful information to develop use cases. As indicated previously, the use cases discovery and description can be improved starting from a more detailed observation of the organizational models as well as of improvement of the proposed heuristics in this work. However, it is important to notice that the proposal presented in this paper is more complete and systematic than a previous proposal in [24]. The main evolution is concerned with the use of goal oriented analysis how the guide technique to mapping i* to use cases. Our last version of the proposed guidelines help Requirements Engineers to discover goals associated with use cases considering the goals, resources, soft-goals and tasks in i* of the view point of how these elements can represent or be associated with goals of the intended software system.

Finally, we have also consider to discover new use cases, the dependences in the SD model between i* actors and a specific actor that represent an intended software system or part of it. In the next section, we present the final considerations of this paper.

5. Conclusion

In this paper we presented some heuristics seeking to show the viability of integrating organizational models developed using the i* framework with Use Cases in UML. Both techniques were described and the proposed guidelines were applied partially to a meeting scheduler system case study. Starting from the case study it was possible to observe that the existent information in the strategic dependency model as well as in the strategic rationale model can be used as base for the use cases development. Besides, it is enable that the requirement engineers, starting from a more detailed observation of the organizational models, choose the best alternative for the software development as well as concentrate in the use cases that really fulfils the organizational goals. In the traditional development, use cases and scenarios do not consider in an effective way, motivations, intentions and alternatives for the systems development. Now, the use of organizational models can assist these activities.

Using our proposal that integrates Organizational Models and Use Cases, some important issues such as how the system fulfils the organization goals, why it is necessary, what are the possible alternatives, what are the implications to the involved parts, can be better treated and proposed solutions incorporated into the software system development.

Some related works include the proposal of requirements-driven development presented in the ***Tropos*** framework [6] [19] and the proposal of the i* and pUML diagrams integration [1]. These works argue that organizational models are fundamental to develop a more reliable software, which can satisfy the real needs of users and organizations. Some of the future works include:

- Improvement of the proposed guidelines described in this work. The main concerns are: (i) to describe more systematic guidelines, that can aid requirement engineers to relate non-functional requirements [8], which can be derived from sofgoals dependencies in i*, with functional requirements of the system, described through use cases in UML; (ii) to incorporate in the integration/mapping process of the i* and Use cases, the structuring mechanisms of the actors in the i* (agent, role and position).

- Study and analyze of the Tropos framework [6] [7] [19] seeking to detect solutions presented in that project, which can contribute and improve our proposal;
- To extend our proposal to other scenario-based techniques presented in [17] and in CREWS project [22] [23];
- Study other goal-oriented modeling approaches as a way of scenario discovery and description [11] [21] [23] [28];
- Definition of heuristics that aids requirement engineers to elaborate more effective i* organizational models, having the scenarios development to elicit, analyze and document system requirements;

References

[1] Alencar, F., Castro, J., Cysneiros, G., Mylopoulos, J., "From Early Requirements Modeled by i* Technique to Later Requirements Modeled in Precise UML", In Proceedings of the "III Workshop de Engenharia de Requisitos", pp 92-108, Rio de Janeiro, (2000).

[2] Booch, G., Jacobson, I., Rumbaugh, J., "The Unified Modeling Language User Guide" , Addison-Wesley (1999).

[3] Brooks, Frederick P., "No silver bullet; essence and accidents of software engineering", Computer, Vol. 20, No. 4, IEEE, pp. 10-19, April, (1987).

[4] Bubenko, J. A., Extending The Scope of Information Modeling, Proc. 4th Int. Workshop on the Deductive Approach to Information Systems and DataBases, Lloret-Costa Brava, Catalonia, Sept. 20-22, 1993, pp73-98.

[5] Bubenko , J, A., Kirikowa, M., "Worlds" in Requirements Acquisition and Modeling, Sweden, (1994).

[6] Castro, J., Kolp, M. and Mylopoulos, J., Developing Agent-Oriented Information Systems for the Enterprise, Proceedings of the Second International Conference on Enterprise Information Systems (ICEIS00), Stafford, UK, July, (2000).

[7] Castro, Jaelson F., Kolp, M., Mylopoulos, J., A Requirements-Driven Development Methodology In: CAISE'01, The 13Th Conference on Advanced Information Systems Engineering, 2001, Interlaken, Suiça. Proceedings of the 13th Conference on Advanced Information Systems Engineering (in Press). Heildelberg, Alemanha: Springer Lecture Notes in Computer Science, 2001.

[8] Chung, L., Nixon, B.A.,Yu, E., Mylopoulos, J., Non-Functional Requirements in Software Engineering (Monograph), Kluwer Academic Publishers, 472 pp, (2000).

[9] Clarke, Edmund M., Jeanette M. Wing et al. Formal Methods: State of the art and future directions. ACM Computing Surveys n. 4, vol. 28. 1996.

[10] Cockburn, A., Writing Effective Use Cases, (Pre-Pub. Draft #3), Humans and Technology, Addison-Wesley, (2000).

[11] Dardene,A., Lamsweerde, V., Fikas, S., Goal-Directed Requirements Acquisition. Science of Computer Programming, 20, pp. 3-50, 1993.

[12] D'Souza , Desmond F., A.C. Wills, Objects, Components and Frameworks with UML: The Catalysis Approach, Addison-Wesley, November, (1998).

[13] DeMarco, T., Structured Analysis and System Specification, Englewood Cliffs, New Jersey: Prentice-Hall, 1979.

[14] Eriksson, Hans-Erik., Penker, Magnus., Business Modeling with UML: business patterns a work, John Wiley & Sons, 2000.

[15] Jacobson, I., Booch, G., Rumbaugh, J., The Unified Software Development Process, Addison-Wesley (1999).

[16] Jacobson, I., Object Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley (1995).

[17] Leite, J.C.S.P, Rossi, G., Balaguer, F., Maiorana, V., Enhancing a requirements baseline with scenarios. In Proceedings of the Third IEEE International Symposium on Requirements Engineering – RE97, pages 44-53. IEEE Computer Society Press, January, 1997.

[18] Mylopoulos, J., Chung, L., and Yu, E., “From Object-Oriented to Goal-Oriented Requirements Analysis,” Communications of the ACM, 42(1), pp. 31-37. January 1999.

[19] Mylopoulos, J., Castro, J., “Tropos: A Framework for Requirements-Driven Software Development” , Brinkkemper, J. and Solvberh, A. (eds), Information Systems Engineering: State of Art and Research Themes, Lectures Notes in Computer Science, Springer-Verlag, June 2000.

[20] Mylopoulos, J., Chung, L., Wang, H.Q. , Liao, S., Yu, E., `Extending Object-Oriented Analysis to Explore Alternatives' IEEE Software.

[21] Potts, C., Fitness for Use: the System Quality that Matters Most, Proc. Third Int’l Workshop Requirements Engineering: Foundations of Software Quality REFSQ’97, pp. 15-28, Barcelona, June, 1997.

[22] Ralyté, Jolita., Rolland, C., Plihon, V., *Method Enhancement With Scenario Based Techniques*, To appear in *Proceedings* of CAISE 99, 11th Conference on Advanced Information Systems Enguneering Heidelberg, Germany, June 14-18, 1999, (CREWS Report Series 99-10).

[23] Rolland, C., Souveyet, C., Achour, C. B., *Guiding Goal Modeling Using Scenarios*, IEE Transactions on Software Engineering, Vol 24, No 12, Special Issue on Scenario Management, December, 1998.

[24] Santander, V. F. A., Castro, J. B., Desenvolvendo Use Cases a Partir de Modelagem Organizacional, III Workshop de Engenharia de Requisitos (WER) , Rio de Janeiro, 13-14 de Julho, 2000.

[25] Schneider, G., Winters, J. P., *Applying Use Cases: a practical guide*, Addison Wesley, (1998).

[26] Sommerville, Ian., Peter Sawyer, *Requirements Engineering: A good practice guide*, John Wiley & Sons, 1997, ISBN: 0 471 97444 7.

[27] Yu, Eric, *Modelling Strategic Relationships for Process Reengineering,* Phd Thesis, University of Toronto, (1995).

[28] Yu, E., “Why Agent-Oriented Requirements Engineering”, Proc. Third International Wokshop Requirements Engineering: Foundations of Software Quality REFSQ’97, pp 171-183, Barcelona, June, 1997.